Q29 : 不使用 */% 两数相除 只能只用int

1. 使用位移运算 << >> 来进行二的倍数的乘除。这样就避免了 使用 * / %
2. 我们 若 a > 2*2b 那么久可以确定 a > 4 , 若a < 2*2*b 久可以确定 a < 8
3. 由4 < a/b <8 那么必然有a/b = 4 + (a-4b)/b
4. 由于负数最小值容易造成越界，我们将两个数都先变为负数计算,这样上边的负号都要反向。

```
 public int divide(int dividend, int divisor) {
         if(dividend == 0) return 0;
         if(divisor == 1) return dividend;
         if(divisor == -1) {
         if(dividend == Integer.MIN_VALUE) return Integer.MAX_VALUE;
                 return -dividend;
         }
         boolean isneg = dividend < 0 && divisor > 0 || divisor < 0 &&
dividend > 0;
         if(dividend > 0) dividend = -dividend;
         if(divisor > 0) divisor = -divisor;
         int ans = div(dividend, divisor);
         if(isneg) return -ans;
         return ans;
 }
 // a:被除数  b:除数   都是负数。   保证不适用long，且中间不会出现越界的数出现。
 int div(int a, int b){
         if(a > b) return 0;
         // 负数 符号要变
         int num = 1;
         int tb = b;
         while((a >> 1) < tb){
         // 如果对b翻倍，就可能越界， 所以算则 a减半。不能用 <= 号  由于 a>>1可能
牺牲精度，本来是 tb*2不满足的情况变满足
                 num <<= 1;
                 tb <<= 1;
                 // 负数：由于 a/2 < tb  而a没有越界（负数边界） 所以 2*tb也不会
越界
         }
         return num + div(a-tb, b);
 }
```

补码

1. 负数以补码表示。 正数的补码就是自身
2. .>>> 无符号右移： 将数看作是无符号数， 右移时高位补0.

> 若原值是负数，第一次相当于先变为正数，之后每次都相当于 / 2

3. .>> 有符号右移： 将数看作是有符号数，右移动时补符号位。是1补1，是0补0 。不管正负其值相当于 / 2

> 溢出的结果是，32位1，或者32位0.

4. .<< 无符号左移： 左移动时相当于无符号左移动，32位整体左移，

> 若负数次高位是0，则左移会吞掉符号
>
> 每次都相当于 * 2 ， 当越来越大时，会溢出，导致变正数。

Q89 : 格雷编码

1. 二进制转 格雷码 二进制：格雷码 = i : i^(i>>1)
2. 格雷码转 二进制： 最高位相同。 B[i-1] = G[i-1]^B[i] :所以要对二进制先求高位。然后递归的去求低位。

Q136 : 只出现一次的数字： 只有一个数只出现一次，其余数都出现两次。

1. 用亦或运算 a^a = 0. 亦或运算可以做交换率， a^b^c^a^d^b^c = a^a^b^b^c^c^d = d

Q166 : 分数到小数 numerator = 2, denominator = 3 输出 "0.(6)" 表示6循环。

1. numerator = 1, denominator = 2 输出 "0.5"
2. 难点其实就是对小数部分的处理。我们可以模拟除法的执行。 当被除数小于 除数时，被被除数*10， 然后再求商追加到小数。
3. 对循环小数的处理，例如， 1/3 整数部分得到0 然后1*10 = 10 ， 10/3 = 3,第一个小数就是3.余数为1，这个1已经出现过了。
4. 说明出现了循环小数。循环的部分就是上次出现1的位置到现在的位置。

Q190 : 颠倒二进制位 颠倒给定的 32 位无符号整数的二进制位。 11110000 - > 00001111

1. 假设是8位的数字 abcd efgh 最终应变为 hgfe dcba
2. abcdefgh -> efgh abcd 前4位和后四位互换，通过左移四位，右移四位然后 |再拼接在一起。

3. efgh abcd -> gh ef cd ab 对4位中前后两位，互换。

4. gh ef cd ab -> h g f e d c b a 对2位中前后1位互换。 结束。

```
n = (n >> 16) | (n << 16);
n = ((n & 0xff00ff00) >> 8) | ((n & 0x00ff00ff) << 8);
n = ((n & 0xf0f0f0f0) >> 4) | ((n & 0x0f0f0f0f) << 4);
n = ((n & 0xcccccccc) >> 2) | ((n & 0x33333333) << 2);
n = ((n & 0xaaaaaaaa) >> 1) | ((n & 0x55555555) << 1);
```

Q191：位1的个数： int数字中，1的个数。负数由补码表示，位1也是补码的位数

1. 不断把数字最后一个 1 反转， 将 n 和 n-1做 **与运算** 会将最低位的 1 变成 0

```
int count = 0;
for(; n!=0; ++count){
    n &= n-1;  // 会将n的最后一个1变为0
}
```

Q201：数字范围按位与两个整数 left 和 right ans = right&(right-1)&...&(left)

1. 保留left和right二进制位的公共前缀。即为答案。因为

```
1110100
.....
1111111
ans = 1110000
```

2. 从0100 -> 1111 全部与运算结果一定为0. 因为011 -> 100 进位，后两位一定结果是0， 最高位也有0，所以结果为0.

```
// n一直在缩小,n第一次小于等于的时候就恰好是前缀
while (m < n) {
    // 抹去最右边的 1
    n = n & (n - 1);
}
return n;
```

Q204：计数质数

1. 埃式筛选法 当下标来到3, 那么把3的所有倍数位置全部设置为true(是合数).

2. 遇到k为true就不需要去设置k的倍数，因为k的倍数，一定也是设置k为true的那个数的倍数。

3. 时间 稍微大于 > o(n),空间o(n)

Q279： 完全平方数 13 = 9 + 4 用最少的完全平方数使得其和为 n。

1. 这是一个数论问题，但是数论解法太高深了，看不懂。 所以看作背包问题比较好。

```
        dp[i] = Math.min(dp[i], dp[i - k*k]+1);
```

Q292 : Nim 游戏 return (n%4 != 0);

1. 当给对手留下4块石头时， 自己肯定能获胜。但是如何给对手留下4块呢。

2. 我们反着想，如果对手给我们留下4块，那么我们必然会输。这么想就方便了。

3. 假设有n个，且n%4==0,即n是4的倍数。

4. 由于我们先手只能拿 1,2,3. 那么对手对应总是可以拿 3,2,1. 这样一轮下来，就是n-4,仍然是4的倍数。这样到最后，必然是我们输。

5. 若n不是4的倍数，那么我们起手就可以拿1,2,3剩余m个， 此时m是4的倍数。那么对手就必输。

Q338 : 比特位计数 对于输入num,给出 0 <= i <= num的 每一个i的比特位数。如： 输入 5 返回 [0,1,1,2,1,2]

动态规划

1. 原理： 10 : 1010 8： 1000 10的比特位数 恰好是 8的比特位数+1， 即差最后一位。

2. i&(i-1)： 可以消去最后一位1，这样就可以通过10找到8.

3. 通过这样向前的依赖，就可以通过dp获得每一个数的比特位数。

```
for(int i = 1; i <= num; i++){
        dp[i] = dp[i&(i-1)]+1;
}
```

Q371 : 两整数之和： 不使用加法，计算 a+b

1. a ^ b： 得到的是无进位二进制加法的 结果。

2. (a & b) << 1 : 得到的是二进制加法 的 进位 结果。

3. 例如 5： 101 6：110 101 + 110 -> 011无进位就是 有一个1时，得到1，其余都是0

4. 101 + 110 -> 每一位向前的进位是 后一位相与的结果。有两个1才会进位。

```
6 ^ 5 = 3
(6 & 5) << 1  = 8
6 + 5 = 3 + 8
3 ^ 8 = 11
(3 & 8) << 1 = 0
6 + 5 = 3 + 8 = 11 + 0 = 11
```

Q402：移掉K位数字 可以移掉任意位的数字，如 10200，可以移除1，剩下 0200， 只是输出是把前导0省略，输出 200

1. 特点，1432219： 如果要移除 2位。那么就是移除 43， 移除3位就是移除 432 移除4位就是 4322.
2. 从左到右找第一个，当cur位大于(严格大于)cur+1位时，删除cur位。就一定合算的。
3. 于是可以设计一个单调栈，且可以双端出队的栈。用数组最好。数字长度为Len,删除k位，那么数组最长len.
4. 使用top=0表示栈顶。每确定一个不删，就入栈top++。 确保top>=0 k>0

```
char c
for(int i = 1; i < num.length(); i++){
    c = num.charAt(i);
    while( k > 0 && top >= 0 &&  c < stack[top]){ // top 可能为 -1
           top--;
           k--;
    }
    stack[++top] = c;  // 删除以后再入栈c
}
top -= k;  // 当i==len时出for循环。此时k可能仍然>0.
stack[++top] = c;  // 删除以后再入栈c

假如输入：1432239 那么删除3位就是 432  删除3位还剩 1239 ,由于后边是递增序列。必然不能删除第4位了。
于是出for循环时，k = 1 >0  还需要删除一位。那么删除谁呢?
递增序列，必然删除最大的。即后边的k位。 直接把top-=k即可。
于是输出就是 0~top的字符。
```

Q406：根据身高重建队列 [[7,0],[4,4],[7,1],[5,0],[6,1],[5,2]] i:[hi,ki],把人i排在队列中，使其前边只有ki个人的身高大于等于hi

```
[7,0], [7,1], [6,1], [5,0], [5,2], [4,4]
再一个一个插入。
7,0]
[7,0], [7,1]
[7,0], [6,1], [7,1]
[5,0], [7,0], [6,1], [7,1]
```

```
    [5,0], [7,0], [5,2], [6,1], [7,1]
    [5,0], [7,0], [5,2], [6,1], [4,4], [7,1]

    高个子的先选正确的位置，比如现在队列有 ： [7,0], [7,1]
    此时[6,1]这个人入队，因为身高6小于队列中所有人，所以6不管放在哪，已经入队的人站位是不会变错的。6即使放在第一个，
    7.0,7,1前边也不会多出一个比他们高的人，只有更高的身高，才会导致他们站位错误。
```

Q738 : 单调递增的数字 输出 比N小的最大数k, k的每一位，从左到右是递增的。

1. N = 332 -> 299 N=1234 -> 1234 N=10 -> 9
2. N = 366634 -》 6>3,所以6-1 = 5 -> 366599 6>5 -> 365999 6>5 -> 359999
3. 找到第一个下降坡 的山峰 63的6就是下降坡的山峰。 对6--, 变为5. 然后 i--. 发现 65,6又是山峰，5--
4. 直到 3 < 5 结束。 那么5之后的全部改为 9.

```
        i指向第二个3. 3666 **3** 4
        if (i < strN.length) {
                // 从山峰开始向左削掉山峰，取得第一个小的位置，
                while (i > 0 && strN[i - 1] > strN[i]) {  6>3  6>5 6>5 最后
3 < 5
                        strN[i - 1] -= 1;
                        i -= 1;
                }
                // i指向5
                for (i += 1; i < strN.length; ++i) {
                        strN[i] = '9';
                }
        }
```

Q976 : 角形的最大周长： 两边之和大于第三边。 我们每次让第三边是最大的一条边。[3,2,3,4]

1. 先把数组排序。
2. 让第三遍是最后一个元素，然后如果a[i] < a[i-1]+a[i-2] 那么说明以i为第三边的三角形是不存在的。
3. 因为 a[i-1]+a[i-2] 已经取最大值了，仍然不行。 此时就排除了 a[i]作为第三遍。 此时i--.

Q1018 : 可被 5 整除的二进制前缀 [0,1,1] -》 [true,false,false]

1. [0,1,1,1,1,1] ——》 [true,false,false,false,true,false] 1111： 15 是5的倍数。
2. 注意： 并不是后边 三位是 101才是5的倍数。这个是不成立的。只有在判断是否是2^n次方时取余，取商才有用。

```
for(int e : A){
    sum = ((sum << 1) | e) % 5;
    list.add(sum == 0);
}
对于 1111   111: 7  (7*2 + 1)%5 = (7*2%5 + 1)%5 = (7%5*2 + 1)%5
```

取余公式

```
1. (a + b) % p = (a % p + b % p) % p
2. (a - b) % p = (a % p - b % p) % p
3. (a * b) % p = (a % p * b % p) % p
4. (a^b) % p = ((a % p)^b) % p
```

判断x是否是素数

```
**对6求余数，余数为 1或5的才可能是素数**
 //见过比较经典的思路
private static boolean isP(int num) {
        if(num <= 3) {
             return num > 1;    // 2,3都是素数
        }
        //6*n+2;6*n+3;6*n+4;6*n等都不是素数;可过滤掉2/3的判断
        if(num % 6 != 1 && num % 6 !=5) {
              return false;
        }

        double sqrt = Math.sqrt(num);
        // 只对 6的 倍数 -1 的位置进行判断。
        for (int i = 5; i < sqrt; i += 6) {
                //只变量2类数据num % 6 == 1 num % 6 == 5
                if (num % i == 0 || num % (i + 2) == 0) {
                        return false;
                }
        }
        return true;
}
```